Computing the Integer Points of a Polyhedron and its Application to Parametric Integer Programming

Rui-Juan Jing and Marc Moreno Maza

Overview

▷ Motivations ◁

This work is motivated by the analysis and transformation of for-loop nests, with a focus on the generation of parametric kernels for Graphics Processing Units (GPUs). Not only these computations require quantifier elimination over the integers rather than over the reals, but also parametric integer programming (PIP). In order for the well-known polyhedral model to support non-linear constraints that arise from the presence of parameters, one needs efficient algorithms for performing PIP without making any assumptions on the parameters, unlike the PIP algorithms found in the literature.

▷ Main results ◁

We propose a new algorithm for describing (without enumerating) the integer points of a polyhedral set. Under a mild assumption, this algorithm runs in polynomial time in fixed dimension. We show how this algorithm can be used to support two typical PIP problems.

Computing the Integer Points

▷ Main Procedure ◁

Given a polyhedron $K \subset \mathbb{R}^d$, represented over $\mathbb{Q}$ by a system of m linear inequalities $\mathbf{Ax} \le \mathbf{b}$, the function call IntegerSolve(K) returns a disjoint union $(K_1 \cap \mathbb{Z}^d) \cup \cdots \cup (K_e \cap \mathbb{Z}^d)$, where $K_1, \ldots, K_e$ are "simpler" polyhedra such that $K_i \cap \mathbb{Z}^d \neq \varnothing$ holds, for $1 \le i \le e$. Assuming that $\mathbf{x} = (x_1, \ldots, x_d)$ is ordered, say $x_1 > \cdots > x_d$, the term "simpler" means that, any integer point in the projection of K_i on $x_j > \cdots > x_d$, for $1 < j \le d$, can be extended to at least one integer point of K_i itself. This latter property is similar to that of a regular chain in polynomial system solving.

▷ Example ◁

$$\begin{cases} 2x+3y-4z+3w \le 1 \\ -2x-3y+4z-3w \le -1 \\ -13x-18y+24z-20w \le -1 \\ -26x-40y+54z-39w \le 0 \\ -24x-38y+49z-31w \le 5 \\ 54x+81y-109z+81w \le 2 \end{cases}$$

$$\Downarrow$$

$$\begin{cases} 3t_1-2t_2+t_3 \le 7 \\ -2t_1+2t_2-t_3 \le 12 \\ -4t_1+t_2+3t_3 \le 15 \\ 2t_2-t_3 \le 48 \\ -5t_2+13t_3 \le 67 \\ -t_2 \le -25 \\ 2 \le t_3 \le 17 \end{cases}, \begin{cases} t_1 = 15 \\ t_2 = 27 \\ t_3 = 16 \end{cases}, \begin{cases} t_1 = 18 \\ t_2 = 33 \\ t_3 = 18 \end{cases}, \begin{cases} t_1 = 14 \\ t_2 = 25 \\ t_3 = 15 \end{cases}, \begin{cases} t_1 = 19 \\ t_2 = 50 + t_6 \\ t_3 = 50 + 2t_6 \\ -25 \le t_6 \le -16. \end{cases}$$

▷ Complexity ◁

Under an *assumption*, IntegerSolve(K) runs within $O(m^{2d^2} d^{4d^3} L^{4d^3} \mathsf{LP}(d, m^d d^4 (\log d + \log L)))$ bit operations, where L is the maximum absolute value of a coefficient in either $\mathbf{A}$ or $\mathbf{b}$, and $\mathsf{LP}(d, H)$ is an upper bound for the number of bit operations required for solving a linear program, with total bit size H and with d variables. Thus IntegerSolve(K) is single exponential in d and polynomial in m and L, for a fixed d. Our assumption: during the execution of the function call IntegerSolve(K), for any polyhedral set K', input of a recursive call, each facet of the dark shadow of K' is parallel to a facet of the real shadow of K'. This assumption generally holds in computer program analysis.
IntegerSolve(K) relies on three sub-procedures.

Procedure 1: IntegerNormalize$(\mathbf{Ax} \le \mathbf{b})$
Returns $\mathbf{Ax} \le \mathbf{b}$ if it implies no equalities, otherwise returns $(\mathbf{t}, \mathbf{x} = \mathbf{Pt} + \mathbf{q}, \mathbf{Mt} \le \mathbf{v})$, where

- $\mathbf{t}$ is a new unknown vector,
- $\mathbf{x} = \mathbf{Pt} + \mathbf{q}$ gives the general form of an integer solution of the implied equations,
- $\mathbf{Mt} \le \mathbf{v}$ is obtained by substituting $\mathbf{x} = \mathbf{Pt} + \mathbf{q}$ into $\mathbf{Ax} \le \mathbf{b}$.

Procedure 2: DarkShadow$(\mathbf{Mt} \le \mathbf{v})$
Returns $(\mathbf{t}', \Theta)$ where

- $\mathbf{t}' = \mathbf{t} \setminus t_1$ holds,
- Θ is a linear system in the $\mathbf{t}'$-variables such that any integer point solving of Θ extends to an integer point solving $\mathbf{Mt} \le \mathbf{v}$.

Principle: considering the variable t_1, for any upper bound $l_1 : a_1 t_1 + \mathbf{a't'} \le v_1$ with $a_1 > 0$ and lower bound $l_2 : b_1 t_1 + \mathbf{b't'} \le v_2$ with $b_1 < 0$ do:

$-b_1\mathbf{a't'} + a_1\mathbf{b't'} \le -b_1 v_1 + a_1 v_2 - (a_1 - 1)(-b_1 - 1)$ ← dark projection generated by l_1 and l_2

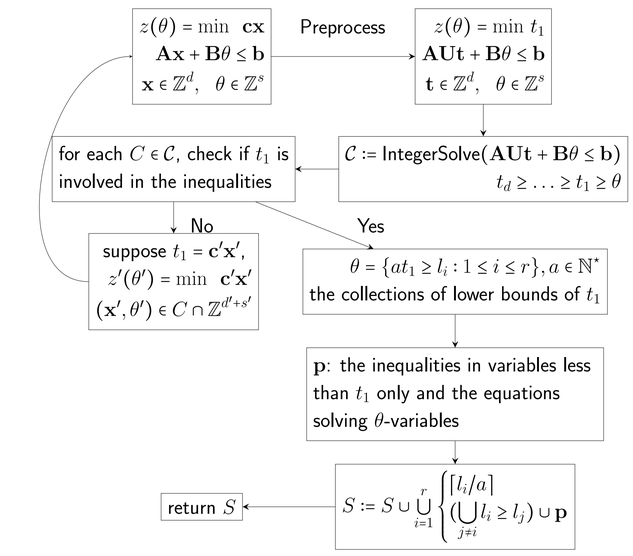
Procedure 3: GreyShadow$(\mathbf{Ax} \le \mathbf{b}, \mathbf{Mt} \le \mathbf{v}, \mathbf{t}', \Theta)$
Let R be the *real shadow* of K, that is, its standard projection on $(x_2, \ldots, x_d)$. Let $d_1, \ldots, d_r$ be the dark projections computed by DarkShadow$(\mathbf{Mt} \le \mathbf{v})$. We call $D := R \cap d_1 \cap \cdots \cap d_r$ and $R \setminus D$ the *dark shadow* and *grey shadow* of K, respectively.
Then, the grey shadow of K has a disjoint decomposition: $R \setminus D = \bigcup_{1 \le i \le t} G_i$, where $G_i = R \cap d_1 \cap \cdots \cap d_{i-1} \cap \neg d_i$ and $\neg d_i$ is the negation of d_i for $1 \le i \le r$.
Principle: considering the variable t_1 again, for any upper bound $l_1 : a_1 t_1 + \mathbf{a't'} \le v_1$ with $a_1 > 0$ and any lower bound $l_2 : b_1 t_1 + \mathbf{b't'} \le v_2$ with $b_1 < 0$ do:

1. let $B := \lfloor (-a_1 b_1 - a_1 + b_1)/a_1 \rfloor$;
2. for i from 0 to B
   - output what IntegerSolve returns when applied to $\{b_1 t_1 + \mathbf{b't'} = v_2 - i\} \cap \mathbf{Mt} \le \mathbf{v} \cap \{-b_1\mathbf{a't'} + a_1\mathbf{b't'} > -b_1 v_1 + a_1 v_2 - (a_1 - 1)(-b_1 - 1)\}$,
   - replace $\mathbf{Mt} \le \mathbf{v}$ by $\mathbf{Mt} \le \mathbf{v}$ augmented with $-b_1\mathbf{a't'} + a_1\mathbf{b't'} \le -b_1 v_1 + a_1 v_2 - (a_1 - 1)(-b_1 - 1)$.

Parametric Integer Programming

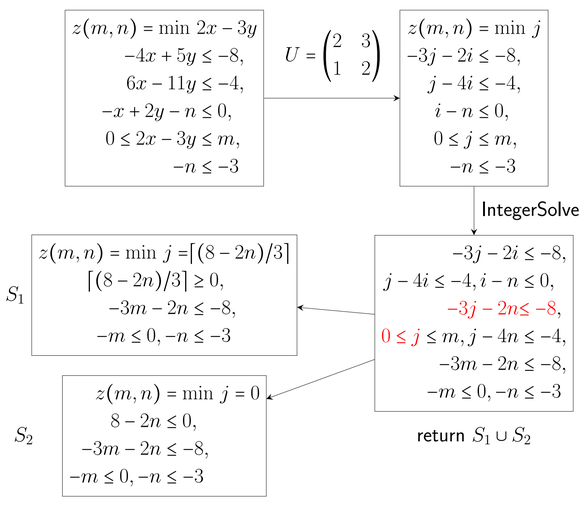
We introduce how to use IntegerSolve solving the parametric integer programming. Note that $\mathbf{A} \in \mathbb{Z}^{m \times d}, \mathbf{B} \in \mathbb{Z}^{m \times s}$, θ is the vector of paremeters. We initialize $S = \{\ \}$.

PIP with an objective function:

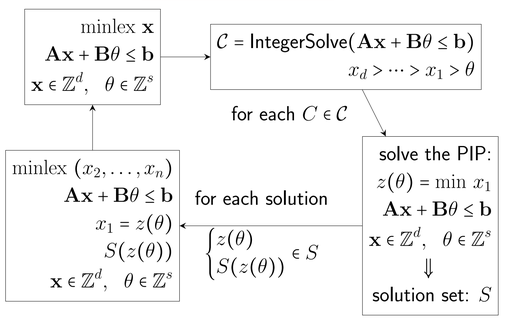


Preprocess: let g be the gcd of elements in $\mathbf{c}$, let $\mathbf{U} \in \mathbb{Z}^{d \times d}$ be a unimodular matrix, $s.t.$ $(g, 0, \ldots, 0) = \mathbf{cU}$

▷ Example ◁



PIP with minlex:



▷ Example ◁

Compute the minlex of (i, j) under the constraints:

$$\begin{cases} 0 \le m - i \le m \\ 0 \le n - j \le n \\ 2(m-i) + (n-j) + k - 2m - n = 0 \\ i \ge 0, j \ge 0 \end{cases}$$

⇓ IntegerSolve

$$\begin{cases} j = -2i + k, -i \le 0, \\ -2i + k - n \le 0, 2i - k \le 0, \\ i - m \le 0, -k \le 0, \\ k - n - 2m \le 0, \\ -m \le 0, -n \le -1 \end{cases} \quad \begin{cases} j = -2i + k, k = 2i, \\ i - m \le 0, n = 0, \\ -i \le 0 \end{cases}$$

$$\Downarrow \qquad \Downarrow$$

$$(i = \lfloor k/2 \rfloor, j = k - 2\lfloor k/2 \rfloor) \quad \begin{cases} \lfloor k/2 \rfloor - m \le 0, \\ n = 0, -\lfloor k/2 \rfloor \le 0 \end{cases}$$

$$\Downarrow$$

$$\min i$$

$$\begin{cases} -i \le 0, \\ -2i + k - n \le 0, 2i - k \le 0, \\ i - m \le 0, -k \le 0, \\ k - n - 2m \le 0, \\ -m \le 0, -n \le -1 \end{cases} \Rightarrow \cdots \begin{cases} (i = 0, j = k) \\ k - n \le 0, -k \le 0, \\ k - n - 2m \le 0, \\ -m \le 0, -n \le -1 \end{cases}$$

$$\Downarrow \quad \vdots$$

$$(i = \lceil \frac{n-k}{2} \rceil, j = k - 2\lceil \frac{n-k}{2} \rceil)$$

$$\begin{cases} k - n \le 0, -2\lceil \frac{n-k}{2} \rceil + k - n \le 0, \\ 2\lceil \frac{n-k}{2} \rceil - k \le 0, \lceil \frac{n-k}{2} \rceil - m \le 0, \\ -k \le 0, k - n - 2m \le 0, -m \le 0, -n \le -1 \end{cases}$$

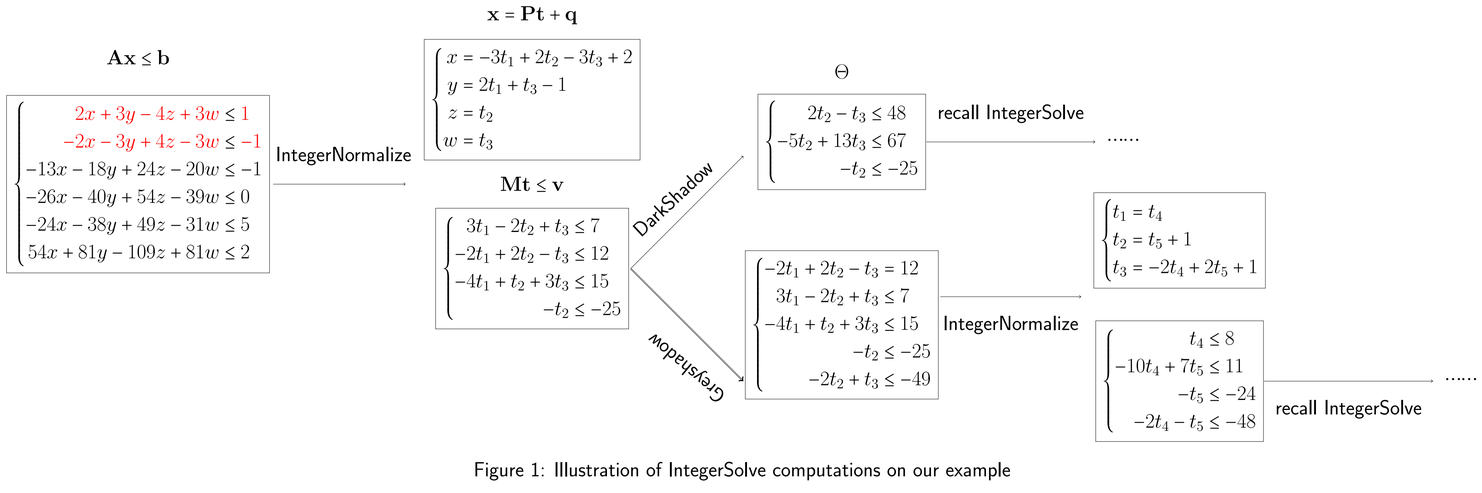


Figure 1: Illustration of IntegerSolve computations on our example

Figure 2: The real, the dark and the grey shadows.

Left-hand side of Figure 2: the polyhedron defined in $\mathbb{R}^3$ by $\mathbf{Mt} \le \mathbf{v}$ together with its dark shadow D (shown in dark blue) as well as its projection on the (t_2, t_3)-plane, denoted by R and called *real shadow* by W. Pugh. Right-hand side of Figure 2: planar view of D and R. This explains why Procedure 3 is needed.